Journal of Computational and Applied Mathematics 40 (1992) 285–296
North-Holland

CAM 1110

On the solution of highly structured nonlinear equations

W.E. Hart
Department of Computer Science, University of Essex, Colchester, United Kingdom

F. Soesianto *
Department of Electrical Engineering, Gadjah Mada University, Yogyakarta, Indonesia

Received 3 August 1990
Revised 6 June 1991

Abstract

Hart, W.E. and F. Soesianto, On the solution of highly structured nonlinear equations, Journal of Computational and Applied Mathematics 40 (1992) 285–296.

We introduce a quasi-Newton update for nonlinear equations which have a Jacobian with sparse triangular factors and consider its application, through an algorithm of Deuflhard, to the solution of boundary value problems by multiple shooting.

Keywords: Quasi-Newton method, boundary value problems, multiple shooting.

1. Quasi-Newton updates

There have been several attempts to integrate the quasi-Newton construction with structural properties of the Jacobian of nonlinear systems which derive from approximations to functional equations [11]. The objectives are to obtain the rapid convergence rates of Newton-like iterative methods simultaneously with a reduction in the computational expense associated with high-dimensional problems. Examples of such methods are the "sparse Broyden" method [3,22] and the Dennis and Marwil method [5], both of which are aimed at general sparse problems. Another such method is that of Kelley and Sachs [17], which is directed more specifically at boundary problems in differential equations. In the first and third of these, approximations are made to the Jacobian and a matrix factorisation is then required for the solution of the linear

Correspondence to: W.E. Hart, Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, United Kingdom.

* This author's research was financially supported by the Government of the Republic of Indonesia World Bank Project XVII.

equations at each iteration. The Denn's and Marwil technique, however, deals with a factorisation of the Jacobian, updating one of the factors in quasi-Newton manner and holding the other constant in modified Newton fashion. An alternate scheme, due to Johnson and Austria [15], updates both U and L^{-1}. Although the Jacobian is sparse, L^{-1} may not be, and under these circumstances this update is relatively expensive. In this paper we propose a method which directly updates both sparse triangular factors, and consider its application to multiple shooting [6–8,10].

The proposed update is a least-change, row-wise procedure constructed through the following elementary lemma.

Lemma. *Let* $a=(a_j)\in\mathbb{R}^N$, $\beta\in\mathbb{R}$ *and* $\sigma\subseteq\{1,2,\ldots,N\}$ *be given. Then if* $\sum_{k\in\sigma}a_k^2\neq 0$, *the minimizer* $\xi=(\xi_j)\in\mathbb{R}^N$ *of* $\|\xi\|_2$ *which satisfies* $a^{\mathrm{T}}\xi=\beta$ *and* $\xi_j=0$, $j\in\bar{\sigma}$, *is*

$$\xi_j=\frac{\beta a_j}{\sum_{k\in\sigma}a_k^2},\quad j\in\sigma. \tag{1}$$

If the nonlinear system to be solved is $F(x)=0$, where $x\in\mathbb{R}^N$, then a single quasi-Newton step, updating an approximation x to $\bar{x}$, is

$$LUs=-F(x), \tag{2a}$$

$$\bar{x}=x+s, \tag{2b}$$

$$\bar{L}=L+M, \tag{2c}$$

$$\bar{U}=U+V. \tag{2d}$$

Here we suppose that LU is the Doolittle factorisation of the approximation B to the Jacobian $F'(x)$. $\bar{L}$ and L are unit lower triangular, and M is strictly lower triangular; all have the same sparsity pattern. $\bar{U}$, U and V are upper triangular with the same sparsity pattern. We seek M and V subject to these conditions and satisfying the quasi-Newton equation

$$\bar{L}\bar{U}s=(L+M)(U+V)s=y=F(\bar{x})-F(x). \tag{3}$$

The update is performed by computing the successive rows of $M+V$ in the following manner [12].

Generally we partition row i of $M+V$ as $[m_i^{\mathrm{T}}\ v_i^{\mathrm{T}}]$, where $m_i\in\mathbb{R}^{i-1}$, $v_i\in\mathbb{R}^{N-i+1}$ and we introduce a new vector $r\in\mathbb{R}^N$ and partition $s^{\mathrm{T}}=[s_i^{\mathrm{T}}\ \bar{s}_i^{\mathrm{T}}]$ and $r^{\mathrm{T}}=[r_i^{\mathrm{T}}\ \bar{r}_i^{\mathrm{T}}]$ similarly. We also introduce compatible partitioning of the ith row of L and U, such that $[l_i^{\mathrm{T}}\ u_i^{\mathrm{T}}]$ represents the ith row of $L+U-I$. Let y_i be the ith component of y.

The update of the first row is immediate, since the first element in $(L+M)(U+V)s=y$ reduces to $u_1^{\mathrm{T}}s+v_1^{\mathrm{T}}s=y_1$, and we then apply the Lemma, with $\beta=y_1-u_1^{\mathrm{T}}s$, and $a=s$ both of which are computable.

Now suppose rows $1,\ldots,i-1$ of M and V have been computed. Then we have available the first $i-1$ elements of the vector $r=(U+V)s$. From element i of the quasi-Newton equation and using the unit lower triangular characteristic of L, we have from (3):

$$m_i^{\mathrm{T}}r_i+v_i^{\mathrm{T}}s_i=y_i-l_i^{\mathrm{T}}r_i-u_i^{\mathrm{T}}s_i.$$

We now use the Lemma, identifying the computable right-hand side with β, and the left-hand side as the inner product $a^{\mathrm{T}}\xi$, where $a^{\mathrm{T}}\equiv[r_i^{\mathrm{T}}\ \bar{s}_i^{\mathrm{T}}]$ is known and $\xi^{\mathrm{T}}=[m_i^{\mathrm{T}}\ v_i^{\mathrm{T}}]$ is the desired update at this ith stage.

Table 0

Computation	Dense Jacobian	Tridiagonal Jacobian
Factorization $B = LU$	$\frac{1}{3}N^3 - \frac{4}{3}N + 1$	$2(N-1)$
Linear equation solving	N^2	$3N-2$
Update operation		
Sparse Broyden method	$2N^2 + N$	$7N-4$
Dennis–Marwil method	$N^2 + N$	$4N-2$
The new method	$\frac{3}{2}N^2 + \frac{3}{2}N - 1$	$6N-4$

We consider three items of computational expense, namely the *factorisation* of B into L and U, the *solution* of the triangular systems $Lr = y$ and $Us = r$ for s, and the *update operations* on the factors. The updating cost, expressed in terms of the number of multiplications and divisions, is $K_1 + K_2 + K_3$, with K_1 the number of nonzero elements to be updated, and K_2 and K_3, respectively, the number of multiplications require to compute the $a^T a$'s and the β's. Since for the general sparse case the formulae are not informative, we consider two extreme cases, namely when the Jacobian $B \in \mathbb{R}^{N \times N}$ has no sparsity and when it is tridiagonal. Table 0 is obtained.

At each iteration step the three methods considered here need to update matrices and solve triangular systems. In addition the sparse Broyden method needs a matrix factorisation and either additional storage for the unfactored matrix or additional computation to recover the matrix from its factors. In the Dennis–Marwil and the new method this factorisation is required only once, after which the unfactored matrix is not required. On the basis of the above operation count, the Dennis–Marwil method is the cheapest of the methods considered here when the Jacobian is banded. Each of the methods exhibits superlinear convergence properties and it is the purpose of the rest of this paper to determine how the differences in the Jacobian approximations affect overall costs in a user algorithm for a specific type of applications.

2. Nonlinear equations in multiple shooting

Given the two-point boundary value problem $u' = f(t, u)$ in $t \in [a, b]$, $r(u(a), u(b)) = 0$ with $u : [a, b] \to \mathbb{R}^n$, $f : [a, b] \times \mathbb{R}^n \to \mathbb{R}^n$, $r : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$. To approximate the solution of this problem by multiple shooting we introduce $m \geqslant 2$ mesh points $a = t_1 < t_2 < t_3 < \cdots < t_m = b$, and approximate $u(t_j)$ by $x_j \in \mathbb{R}^n$, $j = 1, \ldots, m$, which are specified as follows. For $j = 1, \ldots, (m-1)$, we integrate from t_j to t_{j+1} with initial value x_j, the computed value at t_{j+1} is $\tilde{x}_{j+1}$ (say), and the vectors x_j are chosen so that the continuity conditions, $F(x_j, x_{j+1}) \equiv \tilde{x}_{j+1} - x_{j+1} = 0$ are satisfied. Expressing these and the boundary conditions as a system of nonlinear equations in x_j:

$$F_j(x_j, x_{j+1}) = 0, \quad j = 1, \ldots, m-1, \tag{4a}$$

$$F_m \equiv r(x_1, x_m) = 0, \tag{4b}$$

the Jacobian $J(x)$ has the form

$$J(x) - \begin{bmatrix} G_1 & -\bar{G}_2 & & & \\ & G_2 & -\bar{G}_3 & & \\ & & \ddots & & \\ & & & G_{m-1} & -\bar{G}_m \\ A & & & & B \end{bmatrix} \in \mathbb{R}^{N\times N}, \quad N = mn,$$

where, for $j = 1, 2, \ldots, m-1$,

$$G_j = \frac{\partial F_j}{\partial x_j}, \qquad \bar{G}_{j+1} = \frac{\partial F_j}{\partial x_{j+1}} = 1, \qquad A = \frac{\partial r}{\partial x_1}, \qquad B = \frac{\partial r}{\partial x_m}.$$

Solving (4) by Newton's method requires, at each iteration, the solution of a linear system $J(x)s = -F(x)$, $s \equiv [s_1\ s_2\ s_3 \ldots s_m]^T$ and therefore the computation of $J(x)$. The Jacobian is not explicitly available. Approximation based on a finite-difference approximation represents the largest portion of the computing cost. Including the residual norm computation, it amounts to approximately 90% of the total computing time (based on [9, the OVHB entry of Table 1]). By introducing a quasi-Newton method we expect to reduce this cost, and by a direct update to the matrix factors, to reduce costs associated with solving the linear equations. Improvements in overall efficiency will be obtained only if the resulting iterative process converges sufficiently rapidly.

3. BVPSOL

BVPSOL is a multiple shooting algorithm [8] to solve boundary value problems. Shooting is in one direction. The initial-value problem integrator is not specified by Deuflhard; we use a Runge–Kutta–Merson algorithm. BVPSOL uses the damped quasi-Newton method $\bar{x} = x + \lambda B^{-1}F(x)$, where $B = F'(x)$, and chooses the damping factor λ to reduce the natural level function $T(x) = \| B^{-1}F(x) \|_2$.

Convergence is monitored by the monotonicity test $\| D^{-1}\bar{s} \|_2 \leqslant \| D^{-1}s \|_2$, $\| \bar{s} \|_2 = \| B^{-1}F(x + \lambda s) \|_2$, $\| s \|_2 = \| B^{-1}F(x) \|_2$ at some value λ, where, in order to promote invariance under regauging of the components of x_j, BVPSOL introduces a scaling matrix $D = \mathrm{diag}(D_1 D_2, \ldots, D_m)$, $D_j \in \mathbb{R}^n$, $D_i \neq D_j$ computed from x_j. [*Note*: $\bar{s}$ at the end of iteration step i is not equal to s at step $i+1$, which could lead to a cyclic sequence of the natural level function (see [1]), or 'misleading' iteration progress.]

The linear equations are solved using block elimination without pivoting, reducing the system of equations for $[s_1\ s_2\ s_3 \ldots s_m]^T$ to a system for s_1 of the form $Es_1 = -z$, with E and z are computable. Special iterative refinement is used to deal with numerical instability arising from the linear solver used. According to [8], the origin of this instability is the lack of coherence of the errors arising from the computation of z and those from the computation of s_j, $j = 2, \ldots, m$. However, this iterative refinement gives additional information to compute the approximate error tolerance for the integrator. The matrix E is decomposed into QR factors, where Q is

orthogonal and R is upper triangular. This enables a rank reduction strategy to deal with problems of near singularity. Thus the block elimination was chosen with other benefits in mind.

The first iteration is based on a finite-difference approximation B. A rank-1 update on G_j, $j = 1, 2, \ldots, m-1$, is chosen when $\bar{\lambda} \geqslant 2\lambda$, where $\bar{\lambda}$ is a computable a posteriori estimate of λ. If this update fails to make progress, then BVPSOL makes a restart with finite differences.

We have investigated five options, which we call Methods 1–5. For later reference we also call the BVPSOL algorithm Method 0. The normal damping strategy and the restart scheme of BVPSOL are retained. For difficult cases a rank reduction device, implicitly using a pseudo-inverse, is used in BVPSOL to deal with "extremely critical examples" which require $\lambda < \lambda_{\min}$. Since pseudo-inverse computation by LU factors is expensive, we allow the damping factor to assume a minimum value smaller than the value set in BVPSOL (i.e., $\lambda_{\min} = 0.01$). At the expense of a possible increase in the integration cost, we do this in the numerical experiments in order to obtain termination whether the algorithm succeeds or fails. With certain simple forms of separable boundary conditions at $t = a$, BVPSOL avoids unnecessarily computing certain columns of the Jacobian to save some integration cost, but in all our options all columns of the approximate Jacobian are explicitly approximated. Preliminary experiments lead us to introduce a modification to the scaling matrix D, such that $D_i = D_j$ for all i and j, giving faster convergence.

Two linear solvers are added. Linear solver α implements blocked LU factorisation and back-substitution without pivoting and iterative refinement. It is used in Methods 1–3. Linear solver β stores the nonzero elements of the approximate Jacobian in 3 two-dimensional arrays P, Q and R in $\mathbb{R}^{N \times n}$, as in PASVA3 (see [20]). It is based on a combination of row and column interchanges so as to keep the fill-in within the arrays. It is used in Methods 4 and 5.

Method 1 uses *finite differences* to approximate the Jacobian in each iteration step and is used for comparative purposes. Method 2 is similar to Method 0 in the choice of update, but it differs in the linear solver used. Instead of block elimination, it performs LU factorisation. For that reason we also call Method 2 the *modified BVPSOL*. Method 3 performs the proposed LU update on the diagonal block G_j, $j = 1, 2, \ldots, m-1$, but it also takes advantage of the relations in the blocked matrices of the LU factors. Method 4 is *our new method*, implementing the updating on the LU-factored Jacobian matrix. In this respect, Method 3 could be considered as a *variant of our new method*. Finally, Method 5 implements the *Dennis and Marwil* update.

We follow Dennis and Marwil [5] in choosing not to update rows which make the resulting matrix near singular.

4. Some numerical results

Key to the tables

$n \times m$ = order of the differential equation $\times$ number of multiple shooting point nodes. Initial residual = $\| F(x^0) \|_2$. MSHP = number of integration steps in $[a, b]$. FCN calls = number of calls to the ordinary differential equation function. M/D = number of multiplication or division operations to solve the linear equations. The number of iterations to reach convergence is ν, and the work involved is expressed as $i + j + k$ (ν), where i = number of finite-dif-

ference matrix approximation, j = number of rank-1 updates and k = number of LU updates (depending on the method). If $i+j+k > \nu$, updating failed to produce a decrease in the level function at some step, and a finite-difference restart was made. CPU-time is reported in milliseconds. Final residual = $\| F(\bar{x}) \|_2$ when convergence is reached (the termination criterion used is $\| D^{-1}F(\bar{x}) \|_\infty \leqslant \mathrm{TOL}$).

Types of failure

F1: integration failure because overflow occurred when evaluating $\| F(\bar{x}) \|_2$. F2: an insert of new nodes (larger value of m) was requested, because special iterative refinement fails to overcome the problem arising from the lack of coherence. Failures after an update operation are: (i) F3s: due to near singularity, a proper value of λ could not be chosen, or (ii) F3i: cannot evaluate residual either directly after the update or several iteration steps after the update. F4: failure to compute λ as a result of a cyclic sequence of natural level function values. This "misleading" iteration progress is distinguished from false convergence (F5), where the sequence of natural level function values converges to an acceptable termination value at increasing residual norms.

In the experiments, m equally spaced nodes are used throughout.

Example 1 (Hart and Soul [13]).

$$u'' + 3(u')^2 + u^3 = t^6 + 12t^2 + 2, \quad \text{in } [-1, +1].$$

The solution is $u(t) = t^2$. Initial estimate is obtained using the formula $u^0(t) = (t + 0.13)^2$, $u'^0(t) = 2(t - 0.13)$; $m = 9$, 10, 11 and 17. See Table 1.

For $m = 9$, Method 0 failed to converge, Methods 1, 2 and 4 returned F5, and Methods 3 and 5 returned F3s. For $m < 9$ all methods reported F1, since initial residual is high ($\sim 10^{+5}$).

Example 2 (Roberts [21]).

$$\epsilon u'' = u^3, \quad \text{in } [0, 1], \qquad u(0) = 1, \quad u(1) = 2.$$

We take $\epsilon = 0.001$ when there is a boundary layer at both end points. The explicit solution is unknown. Initial estimates were derived from $u^0(t) = 1$, $u'^0(t) = 0$. See Table 2. Methods 1–3 suffered F1 failure at the first iteration. Method 5 reported F1 directly after the update.

Example 3 (Lastman [18]).

$$\begin{aligned}
x_1' &= x_2, & x_1(0) &= -5,\\
x_2' &= -x_1 + (1.4 - cx_2^2)x_2 - 8x_4, & x_2(0) &= -5,\\
x_3' &= x_4 - 2x_1, & x_3(2.5) &= 0,\\
x_4' &= -x_3 - x_4(1.4 - 3cx_2^2), & x_4(2.5) &= 0,
\end{aligned}$$

where $c = 0.14$. Exact solution is unknown. For initial estimate, we use $x_1^0(t) = x_2^0(t) = -5$, $x_3^0(t) = 2$, $x_4^0(t) = 0$. Lastman reports convergence in 15 iterations. See Table 3. Methods 1 and 2 returned F4 failure when $n \times m = 4 \times 51$. For $m < 17$, all methods reported F1 failure. Methods 3 and 5 returned F3i or F3s after update, except for $m = 101$.

Table 1

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
2×10	0	44	8864	2716	3+4+0 (7)	$0.199 \cdot 10^{-5}$	3099
10.4442	1	44	11533	2156	6+0+0 (6)	$0.235 \cdot 10^{-6}$	3998
	2	43	9464	2514	3+4+0 (7)	$0.199 \cdot 10^{-5}$	3337
	3	43	21552	5977	9+0+6 (11)	$0.501 \cdot 10^{-6}$	7565
	4	44	10690	1408	4+0+4 (8)	$0.979 \cdot 10^{-6}$	3733
	5	F3s at iteration 7 (third update)					
2×11	0	46	8367	3004	3+4+0 (7)	$0.618 \cdot 10^{-7}$	2921
5.4235	1	46	9853	1985	5+0+0 (5)	$0.344 \cdot 10^{-6}$	3423
	2	45	8769	2778	3+4+0 (7)	$0.114 \cdot 10^{-6}$	3130
	3	46	17968	5154	8+0+4 (9)	$0.103 \cdot 10^{-6}$	6331
	4	46	8941	1317	3+0+4 (7)	$0.174 \cdot 10^{-4}$	3184
	5	45	12945	1971	6+0+2 (6)	$0.193 \cdot 10^{-6}$	4560
2×17	0	50	6372	4452	1+5+0 (6)	$0.528 \cdot 10^{-5}$	2345
1.64547	1	51	8841	2501	4+0+0 (4)	$0.224 \cdot 10^{-7}$	3158
	2	50	6482	3750	1+5+0 (6)	$0.528 \cdot 10^{-5}$	2395
	3	F3i at iteration 10 (fifth update)					
	4	52	7198	1484	2+0+3 (5)	$0.790 \cdot 10^{-5}$	2676
	5	F3s at iteration 2 after update					
2×21	0	57	7148	5540	1+5+0 (6)	$0.125 \cdot 10^{-5}$	2666
1.23029	1	56	9809	3109	4+0+0 (4)	$0.223 \cdot 10^{-8}$	3521
	2	55	7185	4667	1+5+0 (6)	$0.125 \cdot 10^{-5}$	2698
	3	54	17383	7556	6+0+3 (6)	$0.644 \cdot 10^{-6}$	6551
	4	56	7896	1848	2+0+3 (5)	$0.548 \cdot 10^{-6}$	2970
	5	56	15968	3926	5+0+3 (7)	$0.728 \cdot 10^{-7}$	5799

Table 2

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
2×26	0	286	100164	14112	7+5+0 (12)	$0.526 \cdot 10^{-5}$	29845
638.398	4	243	97439	7329	7+0+8 (15)	$0.504 \cdot 10^{-5}$	29233
2×51	0	257	76155	18576	6+3+0 (9)	$0.453 \cdot 10^{-5}$	23234
175.731	4	259	83440	11820	6+0+6 (12)	$0.477 \cdot 10^{-4}$	25843
2×101	0	322	102332	36764	6+3+0 (9)	$0.342 \cdot 10^{-6}$	32182
105.240	4	325	108799	22204	6+0+5 (11)	$0.162 \cdot 10^{-5}$	35005

Example 4 (Graney [10], Holt [14], Troesch [24]).

$$
\begin{aligned}
x_1' &= x_2, & x_1(0) &= 0,\\
x_2' &= x_3, & x_2(0) &= 0,\ x_2(b) = 0,\\
x_3' &= -1.55x_1x_3 + 0.1x_2^2 + 0.2x_2 - x_4^2 + 1, & &\\
x_4' &= x_5, & x_4(0) &= 0,\ x_4(b) = 1,\\
x_5' &= 1.1x_2x_4 - 1.55x_1x_5 + 0.2x_4 - 0.2. & &
\end{aligned}
$$

Table 3

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
4×17	0	121	35081	29715	6+3+0 (9)	$0.662 \cdot 10^{-6}$	17673
10.8220	1	120	41944	31246	8+0+0 (8)	$0.185 \cdot 10^{-6}$	20805
	2	119	36296	34922	6+3+0 (9)	$0.674 \cdot 10^{-6}$	18215
	4	118	37951	16791	6+0+5 (11)	$0.135 \cdot 10^{-5}$	19289
4×26	0	122	47385	59125	6+5+0 (11)	$0.162 \cdot 10^{-6}$	24659
8.8749	1	122	65289	63008	10+0+0 (10)	$0.713 \cdot 10^{-6}$	32975
	2	126	48242	68736	6+5+0 (11)	$0.166 \cdot 10^{-6}$	24727
	4	121	47998	26979	6+0+5 (11)	$0.154 \cdot 10^{-5}$	25215
4×51	0	148	83659	142546	8+6+0 (14)	$0.447 \cdot 10^{-7}$	45173
6.6788	4	150	113974	93481	11+0+8 (19)	$0.261 \cdot 10^{-6}$	63106
4×101	0	230	109526	198618	7+3+0 (10)	$0.246 \cdot 10^{-6}$	61176
5.07931	1	227	126278	216866	9+0+0 (9)	$0.659 \cdot 10^{-7}$	68447
	2	227	109852	239694	7+3+0 (10)	$0.253 \cdot 10^{-6}$	61124
	4	235	112466	113015	7+0+4 (11)	$0.372 \cdot 10^{-6}$	65473
	5	225	148178	148733	10+0+2 (11)	$0.132 \cdot 10^{-6}$	85999

We choose $b = 20$. Exact solution is unknown. Initial estimate is derived from $x_1^0(t) = -1$, $x_2^0(t) = x_3^0(t) = x_5^0(t) = 0$, $x_4^0(t) = 1$. See Table 4. Our computed solution agrees with [10]. Method 3 returned F3s, Method 5 (except for $m = 65$) reported F1.

Example 5 (*Minimal Surface Problem* (Concus [4]).

$$\left(1 + u_y^2\right)u_{xx} - 2u_x u_y u_{xy} + \left(1 + u_x^2\right)u_{yy} = 0, \quad \text{in } \Omega\colon 0 < x,\ y < 1.$$

Table 4

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
4×17	0	126	28164	69174	3+7+0 (10)	$0.446 \cdot 10^{-5}$	17617
2.2665	1	123	45173	63432	8+0+0 (8)	$0.539 \cdot 10^{-6}$	27159
	2	125	26107	76764	3+7+0 (10)	$0.433 \cdot 10^{-5}$	16583
	4	125	37110	22982	5+0+4 (9)	$0.432 \cdot 10^{-5}$	22759
4×33	0	139	21253	80891	3+4+0 (7)	$0.256 \cdot 10^{-6}$	14553
1.6165	1	137	29172	69206	5+0+0 (5)	$0.180 \cdot 10^{-6}$	18487
	2	139	21967	98194	3+4+0 (7)	$0.276 \cdot 10^{-6}$	15053
	4	139	21981	31186	3+0+4 (7)	$0.265 \cdot 10^{-5}$	15038
4×65	0	187	30883	162135	3+4+0 (7)	$0.175 \cdot 10^{-6}$	22755
1.4610	1	191	42349	137858	5+0+0 (5)	$0.555 \cdot 10^{-7}$	27980
	2	196	31325	95649	3+4+0 (7)	$0.170 \cdot 10^{-6}$	22869
	4	194	31303	61524	3+0+4 (7)	$0.520 \cdot 10^{-6}$	23079
	5	195	88129	155643	9+0+3 (9)	$0.556 \cdot 10^{-7}$	63687

Table 5

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
6×26	0	90	8364	42823	1+2+0 (3)	$0.209 \cdot 10^{-3}$	7651
0.09312	1	88	17676	51063	3+0+0 (3)	$0.154 \cdot 10^{-4}$	14727
	2	89	8421	51122	1+2+0 (3)	$0.209 \cdot 10^{-3}$	7421
	4	87	8426	15169	1+0+2 (3)	$0.198 \cdot 10^{-3}$	7544
10×26	0	106	12664	163883	1+2+0 (3)	$0.823 \cdot 10^{-3}$	17165
0.17129	1	104	29866	208985	3+0+0 (3)	$0.154 \cdot 10^{-4}$	37938
	2	97	13625	279119	1+3+0 (4)	$0.605 \cdot 10^{-3}$	21738
	4	111	12832	50881	1+0+2 (3)	$0.733 \cdot 10^{-3}$	16721
14×26	0	F2: insert new nodes					
0.28668	1	120	72090	900142	5+0+0 (5)	$0.155 \cdot 10^{-5}$	125967
	2	118	33097	898103	2+3+0 (5)	$0.649 \cdot 10^{-4}$	70441
	4	120	18861	135636	1+0+3 (4)	$0.264 \cdot 10^{-3}$	32712
18×26	0	F2: insert new nodes					
0.44293	1	F3: iter = 2, could not find proper λ					
	2	F3: iter = 2, could not find proper λ					
	4	139	26311	287319	1+0+4 (5)	$0.551 \cdot 10^{-3}$	57105
22×26	0	F2: insert new nodes					
0.66022	1	F3: iter = 1, could not find proper λ					
	2	F3: iter = 1, could not find proper λ					
	4	168	35277	523321	1+0+5 (6)	$0.836 \cdot 10^{-3}$	91858
26×26	0	F2: insert new nodes					
0.93184	1	F3: iter = 1, could not find proper λ					
	2	F3: iter = 1, could not find proper λ					
	4	186	76778	1198887	2+0+4 (6)	$0.108 \cdot 10^{-3}$	230408

The condition on the boundary of Ω is $u(x, y) = (\cosh^2 y - x^2)^{1/2}$, which is the solution. We approximate this problem by a method of lines. We introduce $l \geqslant 1$ lines parallel to the x-axis, such that equally spaced nodes $v_0, v_1, v_2, \ldots, v_{l+1}$ are created on the y-axis boundary of Ω; $h \equiv v_{j+1} - v_j$, $j = 0, \ldots, l$. Now, on line j we introduce $u_y(x) \approx (v_{j+1} - v_{j-1})/(2h)$ and $u_{yy}(x) \approx (v_{j+1} - 2v_j + v_{j-1})/h^2$, $j = 1, \ldots, l$. The u_x and u_{xx} become ordinary derivatives; $u_{xy} \approx u'_y(x) = v'_{j+1} - v'_{j-1})/(2h)$, with $(\cdot)'$ denoting the derivative with respect to x. A boundary value problem in ordinary differential equations in v of order $n = 2l$ is obtained, and used as the model to compute the approximate solution to the partial differential equation (see [16,19]). Table 5 is our result for $m = 26$.

At $n = 18$ (9 lines) the residual norm became large ($\sim 10^5$), so that methods based on linear solver α failed. See Table 6 for other results. We note the persistent failure of Method 0.

5. Discussion

The criteria used to evaluate the performance are: (1) the convergence behaviour (success or failure), (2) the number of iterations required, (3) the number of FCN calls to the ordinary

Table 6

$n \times m$ Initial residual	Method	MSHP	FCN calls	M/D	Number of iterations	Final residual	CPU-time (msec)
8×31	0	109	14642	218170	1+5+0 (6)	$0.638 \cdot 10^{-5}$	18083
0.114	1	104	26748	133675	3+0+0 (3)	$0.169 \cdot 10^{-4}$	28067
	2	101	12025	142898	1+2+0 (3)	$0.344 \cdot 10^{-3}$	14193
	4	107	12023	35442	1+0+2 (3)	$0.313 \cdot 10^{-3}$	13358
	5	107	30540	91691	3+0+2 (5)	$0.145 \cdot 10^{-4}$	33944
10×31	0	F2: insert new nodes					
0.152	1	113	34164	247946	3+0+0 (3)	$0.184 \cdot 10^{-4}$	43842
	2	117	15576	329512	1+3+0 (4)	$0.521 \cdot 10^{-3}$	25232
	4	117	14685	61024	1+0+2 (3)	$0.568 \cdot 10^{-3}$	19465
	5	117	38250	158845	3+0+2 (5)	$0.313 \cdot 10^{-4}$	50602
12×31	0	F2: insert new nodes					
0.197	1	129	42612	417887	3+0+0 (3)	$0.368 \cdot 10^{-4}$	64418
	2	124	19794	695967	1+4+0 (5)	$0.908 \cdot 10^{-3}$	42365
	4	129	17662	95532	1+0+2 (3)	$0.983 \cdot 10^{-3}$	27063
	5	F3: iter = 4, after restart, failed to obtain λ					
14×31	0	F2: insert new nodes					
0.251	1	131	82807	1078330	5+0+0 (5)	$0.810 \cdot 10^{-6}$	145808
	2	136	37779	1077914	2+3+0 (5)	$0.626 \cdot 10^{-4}$	81892
	4	127	20475	141837	1+0+2 (3)	$0.167 \cdot 10^{-2}$	35896
	5	F3: iter = 2, after restart, failed to obtain λ					
16×31	0	F2: insert new nodes					
0.314	1	144	173313	2842482	9+0+0 (9)	$0.403 \cdot 10^{-5}$	349284
	2	147	137403	2842971	7+2+0 (9)	$0.205 \cdot 10^{-4}$	293583
	4	147	24787	226282	1+0+3 (4)	$0.324 \cdot 10^{-3}$	49422
	5	142	64065	528365	3+0+2 (5)	$0.842 \cdot 10^{-4}$	126130

differential equation function (reflecting the total cost of integrations for residual and Jacobian evaluations), and (4) the M/D operations used by the linear solver.

Of particular interest are the BVPSOL algorithm and the modified Method 2 in relation to our new Method 4. The BVPSOL algorithm appears unreliable when applied to problems derived from the method-of-lines. The linear equation solver used appears to be the main source of complication. The modified BVPSOL Method 2 appears to be competitive to BVPSOL for problems of low value of n. We note that Method 3 (the variant of our new method) frequently generates near-singular matrices, requiring additional effort to determine an adequate damping factor, or a finite-difference "restart" of the Jacobian. When this is the case, we frequently observed the "misleading" progress previously mentioned.

The new method appears robust, performing well even under poor initial starting values, and requires similar numbers of iteration steps to the modified BVPSOL method. The Dennis–Marwil Method 5, however, frequently suffers complication resulting from the update operation of the only element in the last row of U. The decision not to update this row leads to poor performance or unreliability. (Solving stiff ODE systems, in [2] is reported that the Dennis–

Marwil method did a poorer job in comparison to implicit updating schemes based on Broyden's method.)

The new update algorithm does produce failure when all the nonzero elements of a Jacobian row coincide with those which are to be ignored by the sparsity. This is common in many quasi-Newton methods. Fortunately under a reasonably chosen imposed sparsity pattern, multiple shooting does produce Jacobians which avoid this difficulty to the proposed method. The Dennis–Marwil update, however, is sensitive to this complication.

The evidence for superlinear convergence of the new method was confirmed in [12] and in tests involving a larger set of 25 boundary value problems (see [23], where a superlinear convergence proof is also given).

If a ranking is to be made on the basis of reliability, the new method is the first choice, followed by the modified BVPSOL and BVPSOL method. On the basis of our evidence, the variant of the new method and the Dennis–Marwil are not recommended.

The CPU-time performance depends, but not exclusively, on the comparative time required to do one FCN call and one M/D operation. One FCN call usually requires more CPU-time than the time to do one M/D. This implies that for low-order boundary value problems, the saving in the floating-point operations is not sufficient to produce significant reduction of the CPU-time. The picture becomes different when we consider method-of-lines related problems, which are characterised by a large number of lines (for accuracy of the model) and a large value of m (for the multiple shooting objectives). For problems of this type, the new method demonstrates that the reduction of the integration cost and floating-point operation counts do materialise into significant reductions in the CPU-time as well.

Acknowledgement

The authors wish to thank the referee for critical comments on the presentation of this paper.

References

[1] U. Ascher and M.R. Osborne, A note on solving nonlinear equations and the natural criterion function, *J. Optim. Theory Appl.* **55** (1987) 147–152.

[2] P.N. Brown, A.C. Hindmarsh and H.F. Walker, Experiments with quasi-Newton methods in solving stiff ODE systems, *SIAM J. Sci. Statist. Comput.* **6** (1985) 297–313.

[3] C.G. Broyden, The convergence of an algorithm for solving sparse nonlinear systems, *Math. Comp.* **25** (1971) 285–294.

[4] P. Concus, Numerical solutions of the minimal surface equations, *Math. Comp.* **21** (1967) 340–350.

[5] J.E. Dennis Jr and E.S. Marwil, Direct secant updates of matrix factorizations, *Math. Comp.* **38** (1982) 459–474.

[6] P. Deuflhard, A relaxation strategy for the modified Newton method, in: R. Bulirsch, A. Oettli and J. Stoer, Eds, *Conference Proceeding on Optimization and Optimal Control* (Springer, Berlin, 1975) 59–73.

[7] P. Deuflhard, A stepsize control for continuation methods with special applications to multiple shooting techniques, *Numer. Math.* **33** (1979) 115–146.

[8] P. Deuflhard, Multiple shooting technique revisited, in: P. Deuflhard and E. Hairer, Eds.,, *Numerical Treatment of Inverse Problems in Differential and Integral Equations* (Birkhäuser, Boston, 1983) 74–95.

[9] H.-J. Diekhoff, P. Lory, H.J. Oberle, H.-J. Pesch, P. Rentrop and R. Seydel, Comparing routines for the numerical solution of initial value problems of ordinary differential equations in multiple shooting, *Numer. Math.* **27** (1977) 449–469.

[10] L. Graney, Range extension applied to the multiple shooting method, Memorandum CSM-18, Dept. Comput. Sci., Univ. Essex, 1977.

[11] A. Griewank, On the iterative solution of differential and integral equations using secant updating techniques, in: A. Iserles and M.J.D. Powell, Eds., *The State of the Art in Numerical Analysis* (Clarendon Press, Oxford, 1986) 299–324.

[12] W.E. Hart, Quasi-Newton methods for sparse non-linear systems, Memorandum CSM-151, Dept. Comput. Sci., Univ. Essex, 1990.

[13] W.E. Hart and S.O.W. Soul, Quasi-Newton methods for discretized non-linear boundary-problems, *J. Inst. Math. Appl.* **11** (1973) 351–359

[14] J.F. Holt, Numerical solution of nonlinear two-point boundary problems by finite difference methods, *Comm. ACM* **1** (1964) 366–373.

[15] G.W. Johnson and N.H. Austria, A quasi-Newton method employing direct secant updates of matrix factorizations, *SIAM J. Numer. Anal.* **20** (1983) 315–325.

[16] D.J. Jones and J.C. South Jr, Application of the method of lines to the solution of elliptic partial differential equations, Report No. 1802, National Research Council, Canada, 1979.

[17] C.T. Kelley and E.W. Sachs, A quasi-Newton method for elliptic boundary value problems, *SIAM J. Numer. Anal.* **24** (1987) 516–531.

[18] G.L. Lastman, Obtaining starting values for the shooting method solution of a class of two-point boundary value problems, *J. Optim. Theory Appl.* **14** (1974) 263–270.

[19] M.N. Mikhail, On the validity and stability of the method of lines and the solution of partial differential equations, *Appl. Math. Comput.* **22** (1987) 89–98.

[20] V. Pereyra, PASVA3: An adaptive finite difference FORTRAN program for first order nonlinear, ordinary boundary-value problems, in: B. Childs, M. Scott, J.W. Daniel, E. Denman and P. Nelson, Eds., *Codes for Boundary-Value Problems in Ordinary Differential Equations* (Springer, Berlin, 1979) 67–88.

[21] S.M. Roberts, On the solution of $\epsilon y'' = y^3$, *J. Optim. Theory Appl.* **50** (1986) 535–541.

[22] L.K. Schubert, Modification of a quasi-Newton method for non-linear equations with a sparse Jacobian, *Math. Comp.* **24** (1970) 27–30.

[23] F. Soesianto, On the solution of highly structured nonlinear equations, Ph.D. Thesis, Univ. Essex, 1991.

[24] B.A. Troesch, A severe test problem for two-point boundary value routines, in: B. Childs, M. Scott, J.W. Daniel, E. Denman and P. Nelson, Eds., *Codes for Boundary-Value Problems in Ordinary Differential Equations* (Springer, Berlin, 1979) 357–363.